Final Project Report: Deep Compression and Capacity Estimation of VGG16

CS 294-082 - Experimental Design for Multimedia Machine Learning (Graduate) - Spring 2019

Varun Murthy (Student Author)
University of California, Berkeley
murthy@berkeley.edu

Dr. Gerald Friedland (Faculty Advisor)
Department of EECS, University of California, Berkeley
fractor@eecs.berkeley.edu

ABSTRACT

The introduction of Deep Neural Networks has revolutionized a variety of tasks ranging from text-to-speech translation with Long Short Term Memory Models (LSTMs) to task decomposition with Modular Neural Networks (MNNs). In the field of image processing, Convolutional Neural Networks (CNNs) have gained prominence because of their ability to exploit locality at different granularities and to hierarchially model features. However, the advantage of this robust classification comes at a significant computational cost.

It is no surprise that model compexity and size is directly proportional to training accuracy. Adding more layers to Deep Neural Networks is very easy, but how do we know when to stop? In this paper we discuss the computational calculations needed to determine how large CNNS really need to be to determine whether they are generalizing or memorizing data. We use the VGG16Net model developed by the Visual Geometry Group at the University of Oxford and repeatedly train it on the MNIST dataset while attempting to reduce its parameters without significantly reducing model accuracy.

KEYWORDS

Compression, Stochastic, Memory-Equivalent Capacity, Generalization, Memorization

1 INTRODUCTION

While Deep Neural Networks have significant use-cases in the tasks mentioned above, their computational overhead prevents them from being easily used in the context of IoT devices, sensor networks, mobile phones & other edge-based embedded systems.

Take AlexNet for example which requires 240MB just to store its weight parameters. VGGNet, while considered an improvement, requires far more at 528MB just to store enough information to classify a single image.

We explore a memory-efficient architecture creation scheme in which VGG16 is parameter pruned via the use of L1 norm, L2 norm, & Reservoir Sampling while measuring the model's Generalization-Capacity ρ, a feature that we define to be the following:

$$\rho = \frac{N}{C}$$

where N is the number of correctly classified points in a testing corpus and C is the model's Memory-Equivalent Capacity (MEC) defined as the minimum number of bits of information needed to capture its details.

, ,
2019.

What follows is the procedure used to stochastically compress a PyTorch instance of VGG16 with pretrained parameters. At each iteration of the compression cycle we use the following methods to prune parameters:

- L1 Norm - $|w - \bar{w}|$
- L2 Norm - $(w - \bar{w})^2$
- Reservoir Sampling - Randomized choice of k neurons to kill from n neurons

We use this pruning to reduce the model's footprint and create a pip-installable Python package for those interested in experimenting with our code.

2 MEMORY-EQUIVALENT CAPACITY CALCULATION

The basis for MEC calculation is discussed in great detail in Gerald Friedland's joint work with Mario Krell & Alfredo Metere published by Lawrence Livermore National Laboratory (September, 2018).

We use the following rules in calculating the MEC of fully connected layers:

- The output of a perceptron is maximally 1 bit
- The maximum memory capacity of a perceptron is the number of parameters in bits (MacKay '03)
- The maximum memory capacity of perceptrons in parallel is additive (MacKay '03 and Friedland '17)
- The maximum memory capacity of a layer of perceptrons depending on a previous layer of perceptrons is limited by the maximum output in bits of the previous layer (Data Processing Inequality, Tishby '12)

We consider the fully-connected architecture of VGG16 for which backpropagation is set to true. In this portion of the model, there is an input layer of 25088 neurons, 3 hidden layers of 4096, 4096 & 256 neurons & an output layer of 10 neurons. Using the above rules, we calculate the MEC to be almost 822 million bits!

3 METHOD

3.1 Experimental Design

We repeatedly trained, tested and squeezed the network on MNIST about 6 times. While training we used an dropout algorithm that we implemented in PyTorch to prevent overfitting. Surprisingly, our accuracy jumped from the high 80s to the low 90s when this was implemented. After each testing period, we calculated the MEC using a custom in-house estimator using the rules described above.

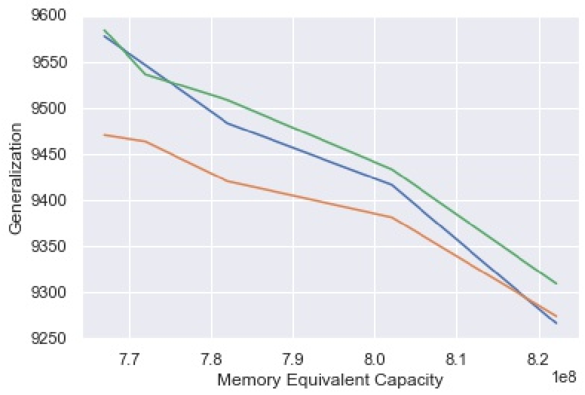

Figure 1: A graph showing the generalization vs. capacity curve of VGG16 trained on MNIST (60,000 training images + 10,000 test images) 3 different times. The vertical axis shows number of correctly classified images and the horizontal axis shows the MEC as a multiple of 10^8.

3.2 Optimization

The following steps were taken to ensure maximum GPU utilization:

- Batch size was kept under 25
- Number of epochs was kept under 6
- CUDA (torch.cuda) was harnessed wherever possible (extreme care was taken to ensure that illegal memory accesses were avoided and leaf variables weren't moved into interiors)
- Model was completely parallelized (torch.nn.DataParallel)

3.3 Implementation

The code was run locally on IPython on a 2018 IBM ThinkPad P72 with Intel i7 8th Gen CPU, 32GB RAM and Nvidia Quadro P600 2GB GPU. Google Colaboratory was also used because of the availability of much powerful GPUs than available locally. TPUs weren't used in the training because the implementation was in PyTorch and not in TensorFlow.

4 CLASS QUESTIONNAIRE

Here we attempt to answer the questions posed towards evaluating machine learners by Professor Friedland. The questions are available on the course website.

Q1: What is the variable the machine learner should predict? What is the required accuracy for success? What impact will adversarial examples have?
The machine learner, in this case VGG16, is trying to predict which numeral from 0-9 the drawn digit represents. There are 10 output classes to choose from. A 99% accuracy is sought after for commercial applications like banking, etc. Adversial examples will try to confuse the machine learner by exploiting the noise present in images to falsely result in a misclassification. e.g. 6 looks like 8

Q2: How much data do we have to train the prediction of the variable? Are the classes balanced? How many modalities could be exploited in the data? Is there temporal information? How much noise are we expecting? Do you expect bias?
The MNIST dataset has 60,000 training images and 10,000 testing images. Each image is 28x28 pixels and each pixel is a 0-225 grayscale value. The classes are balanced and there are no modalities to be exploited (even distribution). We did not expect there to be any bias because of the distribution. Many thanks to Yann LeCun!
Q3: How well is the data annotated (anecdotally)? What is the annotator agreement (measured)?
The data is very well annotated, without a doubt. We use Cohen's kappa to measure the similarity between the two raters here, Yann LeCun and our model.
$k = \frac{1-p_o}{1-p_e}$ where p_o is the accuracy of the model (the number of samples for which the classifications are agreed upon by both labellers) and p_e is the probability of chance agreement which can be considered as an expectation by virtue of its formula: $\frac{1}{N^2}\sum n_{k1}n_{k2}$. In our case, p_o was around 0.95 and $N = 10000$ so although not calculated, our kappa was intuitively rather large.
Q4: Given questions 1-3: Are we reducing information (pattern matching) or do we need to infer information (statistical machine learner)? As a consequence, what seems the best choice for the type of machine learner per modality?
In this classification problem the model is reducing information by applying consecutive convolutions to predict an output class from a set of well-defined classes. Thus, we are in need of a pattern matcher and not a statistical machine learner based on modality as there is no modality to be inferred (even distribution).
Q5: Estimate the memory equivalent capacity needed for the machine learner of your choice. What is the expected generalization? How does the progression look like?
Answered in previous section. Look at Figure 1.
Q6: Train your machine learner for accuracy at memory equivalent capacity. Can you reach near 100% memorization? If not, why (diagnose)?
It is definitely possible to reach near 100% memorization - as stated before it is very easy to add more layers to the network and increase the number of training epochs. CNNs of this scale (AlexNet, SqueezeNet, etc.) have fully-connected architecture with thousands of neurons. If we follow the MEC calculation rules listed earlier, the number of bits can easily be in the range of millions. Consequently, training on a dataset like MNIST or even CIFAR10 with less than 100,000 images won't be an accurate assessment of generalization as we won't be able to see the entire picture - no pun intended - and the generalization-capacity ratio will be very low irrespective of how high our accuracy is.
Q7: Train your machine learner for generalization: Plot the accuracy/ capacity curve. What is the expected accuracy and generalization ratio at the point you decided to stop? Do you need to try a different machine learner (if so, redo from 5)? Should you extract features (if so, redo from 5)?
Answered in previous section. Look at Figure 1.
Q8: How well did your generalization prediction hold on the independent test data? Explain results - how confident are

you in the results?
The model appeared to generalize the independent test data very well. As there were no modalities in the data we were very confident that the machine learner was actually learning the classification of the handwritten digits and not just memorizing the data - this would have been impossible as no two images were the same.
Q9: How do you combine the models of the modalities? Explain your choice. How confident are you in the combination results - does it make sense to combine?
It doesn't make sense to combine different models as there is no visible modality in the data. If there is no skew, no rule is probabilistically favored over another (Bayes' Rule). Thus, mathematically speaking the model has an equal chance to pickup all rules for all data classes.
Q10: What are the final combined results of the system? Are the experiments documented and repeatable (if not, please make sure they are, even for bad results)? Are the experiments reproducible (speculate)?
The experiment is very well documented and is definitely repeatable. We have created a Python package that can be installed via pip. Please visit *www.github.com/Murthy1999/CS294-082* to get started. As for reproducibility, since our experiment involves repeatedly training and squeezing a neural network we can't guarantee the same loss and accuracy metrics but from our experience in this project we estimate that your numbers will be close to ours.

5 REFERENCES

The complete list of references is on the GitHub page.